QECTOR DECODER v3

OFFICIAL USER MANUAL & EXTENDED TECHNICAL REFERENCE

Version 0.6.9 - Quantum Error Correction Decoding Platform

Upstream PyPI Package: qector-decoder-v3==0.6.9 (Released 2026-07-26)
Backend Architecture: Compiled High-Performance Rust Extension (PyO3) + Python 3.9-3.13 Surface
Licensing Model: Source-Available (Free Academic / Personal / Non-Commercial; Commercial License Required)
Primary Zenodo User Manual DOI: 10.5281/zenodo.21363016
Performance Benchmarks DOI: 10.5281/zenodo.21339300 | **Architecture Whitepaper DOI:** 10.5281/zenodo.21320543

Author / Lead Architect: Guillaume Lessard (iD01t Productions)
Author ORCID: 0009-0000-3465-3753
Official Website: https://www.qector.store | **PyPI Portal:** https://pypi.org/project/qector-decoder-v3/
Commercial Licensing Contact: admin@qector.store

Publication Specification Ready for Zenodo DOI Archiving, Google Books & id01t.itch.io Digital Distribution
Document Version Generated: 2026-07-26 12:06 UTC

1. System Introduction & Core Architecture

QECTOR Decoder v3 (v0.6.9) is an enterprise-grade quantum error correction (QEC) decoding library written in Rust and exposed as a zero-overhead Python extension via PyO3. It provides scalable, hardware-accelerated decoding algorithms for fault-tolerant quantum computing architectures, CSS stabilizer codes, and general non-graphlike quantum Low-Density Parity-Check (qLDPC) codes.

The public version string is qector_decoder_v3.__version__ == '0.6.9'. The system provides zero-copy NumPy memory passing, GIL-free native execution threads, and full compatibility with PyMatching, Stim, and Sinter.

Complete Package Layout Tree

```
qector_decoder_v3/
+-- Rust Core Extension (compiled native binary module)
|   +-- Union-Find cluster growing & boundary peeling engines
|   +-- Blossom V perfect matching graph solver & blossom shrink/expand
|   +-- CPU AVX2 parallel SIMD batch decoding pool
|   +-- CUDA GPU batch kernels & memory stream manager
|   +-- OpenCL cross-vendor GPU execution paths
|   +-- Detector Error Model (DEM) graph parser & error collapsing
|
+-- Python Surface Package
    +-- __init__.py          # Public API export & package metadata (__version__ = '0.6.9')
    +-- backend.py           # Native extension bindings & decoder factory routing
    +-- dem.py               # Detector Error Model parsing & hyperedge reduction
    +-- belief_matching.py   # Correlated-noise BP + MWPM decoder (length-n_qubits faithful)
    +-- bposd.py             # Log-domain sum-product BP & OSD-0/1/2 for qLDPC codes
    +-- predecoder.py        # Syndrome pre-filtering & hash table cache
    +-- gnn.py               # GNNBeliefMatcher, GNNPredecoder & GNNTrainer
    +-- codes.py             # Repetition, Ring, Surface, Toy & Bivariate Bicycle code generators
    +-- stim_compat.py       # Stim DEM conversion wrappers (from_stim_detector_error_model)
    +-- sinter_compat.py     # Sinter custom decoder integration (qector_sinter_decoders)
    +-- license.py           # Offline Ed25519 cryptographic token verifier
    +-- rest_api.py          # Local FastAPI / Flask syndrome decoding service
    +-- mcp_server.py        # Embedded JSON-RPC 2.0 Model Context Protocol (MCP) server
    +-- workbench.py         # Multi-process orchestrator & benchmark harness
```

Key Innovations in Release v0.6.9

- **Faithful BeliefMatching Corrections:** BeliefMatching.from_numpy_h returns faithful length-n_qubits correction vectors satisfying H @ c ≡ s (mod 2) on all validated graph-like codes.
- **Exact Log-Domain BP-OSD Engine:** Replaced approximate sum-product BP with exact log-domain sum-product BP by default, extending OSD order support (OSD-0, OSD-1, OSD-2) for dense and qLDPC codes.
- **GNN-Guided Neural Decoding:** Introduced GNNBeliefMatcher, GNNPredecoder, and GNNTrainer for end-to-end deep learning guided MWPM decoding with strict syndrome faithfulness fallbacks.
- **Cryptographically Hardened License Engine:** Ed25519 offline token verifier handles invalid or malformed tokens gracefully without throwing uncaught exceptions; v2 tokens support tier and expiration metadata.
- **Granular Tuning Parameters:** Environment variables QECTOR_BLOSSOM_K_MULT, QECTOR_BLOSSOM_INTRA_PAR, and QECTOR_BLOSSOM_INTRA_THREADS enable fine-grained accuracy vs throughput optimization.
- **Package Stdio MCP Server:** Built-in JSON-RPC 2.0 tool server (run_mcp_server) for direct LLM agent integration with 3 core execution tools.

2. Installation, Platforms & Environmental Options

Pre-compiled binary wheels are hosted on PyPI for 64-bit operating systems. Installation requires Python 3.9 through 3.13.

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

PIP Installation Commands

```
pip install qector-decoder-v3==0.6.9          # Core decoder engine
pip install "qector-decoder-v3[stim]"         # Stim, Sinter, PyMatching, LDPC ecosystem
pip install "qector-decoder-v3[bench]"        # Benchmarking (psutil, matplotlib, scipy)
pip install "qector-decoder-v3[all]"          # Complete quantum research stack
pip install "qector-decoder-v3[dev]"          # Development & testing dependencies
```

Platform & Environment Summary

Item / Parameter	Specification / Value	Operational Function
Python Target	3.9 - 3.13	CPython runtime environment supported
Supported Platforms	Linux x86_64, Windows x64, macOS arm64	Pre-compiled wheels published on PyPI
Licensing Model	Source-available	Free academic / personal / non-commercial; commercial license required for corporate use
Startup Banner Notice	QECTOR_SILENT=1	Set environment variable QECTOR_SILENT=1 to suppress engine initialization banner
Licence Token Variable	QECTOR_LICENSE	Ed25519 token string; overrides tier for testing ('academic' / 'commercial')

Dependency Matrix

Dependency Package	Required Version	Installation Extra	Functional Purpose
numpy	>=1.24.0, <2.3	Core (Default)	Zero-copy syndrome arrays & matrix algebra
cryptography	>=3.4.0	Core (Default)	Offline Ed25519 license token validation
stim	>=1.12.0	extra == 'stim' / 'all'	Circuit-level noise simulation & DEM generation
pymatching	>=2.0.0	extra == 'stim' / 'all'	Reference MWPM benchmark comparison
sinter	>=1.12.0	extra == 'stim' / 'all'	Parallel Monte Carlo sampling orchestrator
ldpc	>=2.0.0	extra == 'stim' / 'all'	BP-OSD reference decoding
beliefmatching	>=0.2, <=2.2.6	extra == 'stim' / 'all'	Belief propagation matching reference
scipy	>=1.10.0	extra == 'bench' / 'all'	Sparse matrix linear algebra & fitting
matplotlib	>=3.7.0	extra == 'bench' / 'all'	Benchmark chart generation
psutil	>=5.9.0	extra == 'bench' / 'all'	Hardware RSS memory profiling
cupy-cuda12x	>=13.0.0	extra == 'cupy-cuda12x'	CuPy GPU array batch decoding
fastapi / uvicorn	>=0.110 / >=0.29	extra == 'dev'	Local REST API web service server
stripe	>=5.0.0	extra == 'dev'	Automated license fulfillment webhooks

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

3. Environment Variables & Performance Tuning Guide

QECTOR Decoder v3 provides environment variable controls to tune execution latency, parallelism, device selection, and matching accuracy:

Variable Name	Default	Affects Results?	Technical Rationale & Operational Behavior
QECTOR_BLOSSOM_K_MULT	2.0	Yes (Accuracy)	Candidate-neighbour search multiplier: k = max(12, ceil(mult * sqrt(n_defects))). Lowering it reduces latency but can exclude the optimal partner on dense instances, producing a heavier (sub-optimal) matching. 2.0 is the tuned minimum that preserved exact-MWPM parity at d >= 15.
QECTOR_BLOSSOM_INTRA_PAR	auto	No (Throughput)	Force intra-decode parallelism for candidate discovery (0=off, 1=on). Unset selects automatically when the graph has >= 64 nodes (d >= 9 for rotated surface codes). Output is bit-identical either way.
QECTOR_BLOSSOM_INTRA_THREADS	unset	No (Throughput)	Size a dedicated Rayon pool for candidate discovery, independent of the global batch pool. Performance only.
QECTOR_CUDA_DEVICE_ID	0	No (Device)	Which CUDA device the native batch / BP-OSD decoders bind to.
QECTOR_OPENCL_DEVICE_ALLOW	unset	Yes (Selection)	Comma-separated substrings matched case-insensitively against OpenCL device names, e.g. nvidia,geforce. Use it to avoid selecting an integrated GPU on multi-device hosts.
QECTOR_SILENT	0	No (Console)	Set to 1 to suppress engine initialization banner and notices.
QECTOR_LICENSE	unset	No (Licensing)	Ed25519 license token string. Overrides tier for testing ('academic' / 'commercial').
QECTOR_DATA_DIR	unset	No (Storage)	Custom directory path for persistent configuration and report exports.

4. Complete Decoder Catalogue (27 Registry Entries)

QECTOR Decoder v3 exposes 27 decoder class implementations, entry points, and integration utilities across multiple API surface layers:

Decoder Module / Class	Best Use / Primary Role	Graph-Like Req.?	Stability Tier
UnionFindDecoder	Low-latency linear-time approximate decoding	Yes (weight <= 2)	Stable
FastUnionFindDecoder	Optimised Union-Find hot-path solver	Yes (weight <= 2)	Stable
BlossomDecoder	Exact MWPM solver (PyMatching parity validation)	No	Stable
SparseBlossomDecoder	Near-optimal low-density perfect matching	Prefer graph-like	Experimental
BeliefMatching	Correlated-noise belief propagation + MWPM	Prefer graph-like	Research
BpOsdDecoder / BPOSDDecoder	qLDPC & general non-graphlike codes (BP+OSD)	No	Experimental
BatchDecoder / CPUBatchDecoder	High-throughput CPU AVX2 parallel batching	Yes (for UF)	Stable
CUDABatchDecoder	NVIDIA GPU hardware accelerated batching	Depends on engine	Runtime-dep.
OpenCLBatchDecoder	Cross-vendor GPU batch decoder	Depends on engine	Build-dep.
AutoDecoder	7-tier resilient self-debugging fallback router	Automated	Stable
HybridDecoder	Union-Find clustering with Blossom routing	Prefer graph-like	Experimental
PredecodedDecoder	Easy-syndrome prefiltering cache	Prefer graph-like	Experimental
LookupTableDecoder	Small distance codes (n_checks <= 20)	No	Experimental

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

Decoder Module / Class	Best Use / Primary Role	Graph-Like Req.?	Stability Tier
SlidingWindowDecoder	Multi-round streaming syndrome decoding	Automated	Experimental
StreamingDecoder	Continuous real-time streaming sessions	Automated	Experimental
GNNBeliefMatcher	End-to-end GNN-guided MWPM decoder	Prefer graph-like	Research
GNNPredecoder / NeuralPredecoder	Learned predecoder front-end for syndrome reduction	No	Research
GNNTrainer	Training harness for GNNPredecoder models	No	Research
DecoderPool	Multi-process CPU/GPU parallel batch pool	No	Stable
get_decoder / clear_decoder_cache	Cached decoder factory interface	No	Stable
decode_mmap	Out-of-core memory-mapped file decoding	No	Stable
DecodeResult / decode_with_diagnostics	Structured decode result container with telemetry	No	Stable
Workbench	High-level orchestration suite	No	Stable
LERBenchmark	Automated Logical Error Rate parameter sweeps	No	Experimental
stim_compat	Stim circuit and DEM conversion helpers	No	Stable Utility
sinter_compat	Sinter custom decoder integration wrappers	No	Stable Utility
rest_api	Local FastAPI decoding web service	No	Local / Review

Important Constraint: Graph-like Rule (UF Family): Every qubit must participate in at most two checks (participation <= 2). Matrices that violate this raise a clear structural error (no silent wrong results). BlossomDecoder, SparseBlossomDecoder, and BpOsdDecoder accept hyperedge / higher-degree codes.

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

5. Code Generators & Hyperedge Workaround Protocol

The package exports instant code generators for standard topological and qLDPC testing:

```
from qector_decoder_v3 import (
    generate_repetition_code_checks,  # (checks, n_qubits)
    generate_ring_code_checks,
    generate_surface_code_checks,     # hyperedge-style surface
    generate_toy_code_checks,
)
```

Generator Name	Topological Structure	Graph-Like Compliant?
generate_repetition_code_checks(d)	Path graph of length d	Yes (Participation = 2)
generate_ring_code_checks(d)	Cycle graph with n_qubits = d^2	Yes (Participation = 2)
generate_surface_code_checks(d)	Hyperedge surface representation	No (d=3 has participation 8)
generate_toy_code_checks()	Small 5-qubit test code	Yes (Participation = 2)
bivariate_bicycle_code(l, m, ...)	qLDPC bivariate bicycle matrix pair (cx, cz)	No (Hyperedge qLDPC)

System Recommendation: For production surface-code work, prefer a Stim DEM (decompose_errors=True) via stim_compat.from_stim_detector_error_model, or use the Workbench's graph-like rotated_surface generator outside this package.

Hyperedge Workaround Function

When using package MCP or when UF-family decoders reject high-degree incidence matrices, use the direct Blossom or BP-OSD interface:

```
import numpy as np
from qector_decoder_v3 import BlossomDecoder, SparseBlossomDecoder, BpOsdDecoder

def decode_hyperedge(checks, n_qubits, syndrome, kind='Blossom', **opts):
    """Use when MCP or UF-family reject high-degree incidence matrices."""
    syn = np.asarray(syndrome, dtype=np.uint8).ravel()
    if kind in ('Blossom', 'blossom'):
        return BlossomDecoder(checks, n_qubits).decode(syn)
    if kind in ('SparseBlossom', 'sparse_blossom'):
        return SparseBlossomDecoder(checks, n_qubits).decode(syn)
    if kind.lower() in ('bposd', 'bp_osd'):
        H = np.zeros((len(checks), n_qubits), dtype=np.uint8)
        for i, c in enumerate(checks):
            for q in c:
                if 0 <= q < n_qubits:
                    H[i, q] ^= 1
        return BpOsdDecoder(H, error_rate=opts.get('error_rate', 0.05),
                            osd_order=opts.get('osd_order', 0)).decode(syn)
    raise ValueError(f'unsupported kind for hyperedge: {kind}')
```

6. Minimal Usage Examples & Production Recipes

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

Single Syndrome (Graph-Like)

```
import numpy as np
from qector_decoder_v3 import UnionFindDecoder, BlossomDecoder

checks, n_qubits = [[0, 1], [1, 2], [2, 3], [3, 4]], 5
syndrome = np.array([0, 1, 0, 0], dtype=np.uint8)

uf = UnionFindDecoder(checks, n_qubits)
print(uf.decode(syndrome))

mwpm = BlossomDecoder(checks, n_qubits)
print(mwpm.decode(syndrome))
```

Batch Decoding

```
from qector_decoder_v3 import BatchDecoder, CUDABatchDecoder
import numpy as np

checks, n_qubits = [[0, 1], [1, 2], [2, 3], [3, 4]], 5
batch = BatchDecoder(checks, n_qubits)
syndromes = np.random.randint(0, 2, (4096, 4), dtype=np.uint8)
corrections = batch.parallel_batch_decode(syndromes)

if CUDABatchDecoder.is_available():
    gpu = CUDABatchDecoder(checks, n_qubits)
    corrections = gpu.batch_decode(syndromes)
```

AutoDecoder - 7-Tier Self-Debugging Fallback

```
from qector_decoder_v3 import AutoDecoder

dec = AutoDecoder(checks, n_qubits)
corrections = dec.batch_decode(syndromes)
# Inspect diagnostics:
print(dec._diag.backend_health)
print(dec._diag.active_backend)
```

BeliefMatching & BP-OSD

```
from qector_decoder_v3 import BeliefMatching, BpOsdDecoder
import numpy as np

H = np.array([[1,1,0,0,0],[0,1,1,0,0],[0,0,1,1,0],[0,0,0,1,1]], dtype=np.uint8)
bm = BeliefMatching.from_numpy_h(H)          # v0.6.9: returns length-n_qubits vector
print(bm.decode(np.array([0,1,0,0], dtype=np.uint8)))

bp = BpOsdDecoder(H, error_rate=0.05, osd_order=0, bp_method='exact')
print(bp.decode(np.array([0,1,0,0], dtype=np.uint8)))
```

Stim Workflow

```
import stim
from qector_decoder_v3 import BlossomDecoder
from qector_decoder_v3.stim_compat import from_stim_detector_error_model

circuit = stim.Circuit.generated(
    'surface_code:rotated_memory_z', distance=5, rounds=5,
    after_clifford_depolarization=0.005,
)
dem = circuit.detector_error_model(decompose_errors=True)
checks, n_qubits = from_stim_detector_error_model(dem)
decoder = BlossomDecoder(checks, n_qubits)
```

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

License Verification

```
from qector_decoder_v3.license import verify_license_token
verify_license_token('')                    # False
verify_license_token('garbage')             # False (no raise; hardened in 0.6.9)
verify_license_token('academic')            # True (dev override)
```

Sinter Integration

```
import sinter
from qector_decoder_v3.sinter_compat import qector_sinter_decoders
# Custom decoders available: qector_belief, qector_blossom, qector_unionfind
custom_decoders = qector_sinter_decoders()
```

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

7. Package MCP Server Specification (run_mcp_server)

```
from qector_decoder_v3 import run_mcp_server
run_mcp_server()   # JSON-RPC 2.0 on stdin/stdout
```

Tool Name	Role / Function	Accepted Decoder Types
decode_syndrome	Single syndrome decoding	UnionFind, FastUnionFind, Blossom, SparseBlossom, bposd, LookupTable, SlidingWindow, Streaming, Auto, Hybrid, BeliefMatching, Predecoded, GNNBeliefMatcher, Batch
benchmark_decoder	Simple latency harness	Same as above
get_decoder_info	Version + partial capability list	None

Important Constraint: Limitation: The package MCP applies a UF-style structural gate (qubit participation <= 2). Hyperedge matrices (including generate_surface_code_checks) are rejected for all decoder_type values with error -32602. There is no argument that disables this gate. Workbench MCP (47 tools) is a separate product surface.

8. Practical Recommendations & Scope Boundaries

Target Research Goal	Recommended Choice
Fast graph-like Monte Carlo	FastUnionFindDecoder / BatchDecoder / AutoDecoder
Exact MWPM validation	BlossomDecoder
Hyperedge / general stabilizer	BlossomDecoder or BpOsdDecoder (direct API)
qLDPC / LDPC	BpOsdDecoder (bp_method='exact', osd_order as needed)
Accuracy research	BeliefMatching / GNNBeliefMatcher
Agent / stdio automation	Package MCP for graph-like only; else direct API + fallback
Licence checks	verify_license_token (offline Ed25519; v2 tokens with tier/exp)

What This Package Is Not

- Not the Workbench application (no 47-tool MCP, no GUI).
- Not a drop-in claim of universal PyMatching replacement (honest latency positioning in docs).
- Not open-source: source-available; commercial use requires a licence.

9. AutoDecoder 7-Tier Fallback Specification (v0.6.8+)

Tier	Backend Engine	Description & Failure Behavior
Tier 1	CUDA batch	GPU batch decoding via NVRTC-compiled kernels
Tier 2	OpenCL batch	Cross-vendor GPU batch decoding
Tier 3	CPU Rayon parallel batch	Multi-threaded parallel CPU batch decoding
Tier 4	CPU batch	Single-threaded CPU batch decoding
Tier 5	CPU single	Per-syndrome CPU decoding
Tier 6	Blossom (exact)	Exact MWPM fallback (guaranteed correctness)
Tier 7	Lookup / pure Python	Pure-Python last-resort fallback tier

System Recommendation: Failed backends are health-scored and suspended; dec._diag.reset_backend_health() re-enables them.

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

10. Validated Claims & Empirical Benchmarks

Package Artefact Validations

- MWPM parity with PyMatching on tested surface distances (LER counts match; latency is workload-dependent; PyMatching often faster on standard MWPM).
- Belief-matching LER reduction on low-distance circuit-level noise (research path; much slower).
- GPU batch bit-identical to CPU when CUDA/OpenCL builds and hardware are available.
- Faithfulness: corrections satisfy H c ≡ s (mod 2) on successful graph-like trials in package tests (including exhaustive repetition d=5).

Aggregated Single-Shot Decoder Ranking (Geometric Mean d=3 to d=19)

Decoder Engine	Geomean Latency (µs)	Geomean Throughput (dec/s)	Distance Coverage
lookup_table	2.28 µs	397,545 dec/s	d=3 to d=5 (n_checks <= 20)
fast_union_find	5.62 µs	169,062 dec/s	d=3 to d=19 (9/9)
union_find	5.79 µs	164,082 dec/s	d=3 to d=19 (9/9)
auto	7.18 µs	133,878 dec/s	d=3 to d=19 (9/9)
blossom	18.74 µs	51,560 dec/s	d=3 to d=19 (9/9)
predecoded	21.59 µs	45,502 dec/s	d=3 to d=19 (9/9)
sparse_blossom	100.90 µs	9,703 dec/s	d=3 to d=19 (9/9)
hybrid	100.90 µs	9,706 dec/s	d=3 to d=19 (9/9)
bp_osd	502.10 µs	1,967 dec/s	d=3 to d=19 (9/9)
auto_router	14,035.30 µs	71 dec/s	d=3 to d=19 (9/9)

CPU (AVX2) vs NVIDIA CUDA GPU Batch Throughput Across All Distances

Distance d	CPU Batch Throughput (shots/s)	CUDA Batch Throughput (shots/s)	GPU Acceleration Factor
d = 3	3,341,548 shots/s	6,140,620 shots/s	1.84x Speedup
d = 5	1,457,301 shots/s	3,842,459 shots/s	2.64x Speedup
d = 7	583,690 shots/s	2,270,727 shots/s	3.89x Speedup
d = 9	382,148 shots/s	1,180,760 shots/s	3.09x Speedup
d = 11	243,784 shots/s	761,644 shots/s	3.12x Speedup
d = 13	175,574 shots/s	483,241 shots/s	2.75x Speedup
d = 15	128,499 shots/s	353,638 shots/s	2.75x Speedup
d = 17	99,470 shots/s	276,936 shots/s	2.78x Speedup
d = 19	79,386 shots/s	224,021 shots/s	2.82x Speedup

MWPM Parity Against PyMatching (Artifact: benchmark_results/stim_ler_d13_d15.md)

Distance d	QECTOR Blossom LER	PyMatching LER	QECTOR Latency (µs/shot)	PyMatching Latency (µs/shot)
d = 13	0.00075	0.00075	820.46 µs	81.12 µs
d = 15	0.00050	0.00050	1965.15 µs	203.20 µs

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753

Belief-Matching Accuracy Improvement (Artifact: benchmark_results/competitive_belief.md)

Distance d	PyMatching LER	QECTOR MWPM LER	QECTOR Belief LER (Correlated Noise)
d = 5	0.00767	0.00767	0.00500 (34.8% LER Reduction)
d = 7	0.00600	0.00600	0.00300 (50.0% LER Reduction)

Native Memory Profile (Distance 13, Batch 16,384 - Artifact: native_memory.json)

Decoder Engine	RSS Memory Delta (MiB)	Memory Footprint Evaluation
fast_union_find	0.02 MiB	Ultra-lightweight footprint
cuda_batch	2.67 MiB	Low host overhead
blossom	5.88 MiB	Exact matching graph structures
cpu_batch	9.41 MiB	Multi-threaded AVX2 buffer allocation

11. Exhaustive PyPI Release History (v0.6.0 - v0.6.9)

Release Version	Release Date	Exhaustive Architectural Features & Patch Notes
v0.6.9	2026-07-26	Faithful BeliefMatching length-n_qubits corrections (H @ c ≡ s mod 2). Exact log-domain BP-OSD sum-product. GNNBeliefMatcher, GNNPredecoder, GNNTrainer. Cryptographically hardened Ed25519 token verifier (malformed tokens return False without exception). Tuning env vars QECTOR_BLOSSOM_K_MULT, intra-thread Rayon pool management, device selection. Package stdio MCP server (run_mcp_server). Dynamic payment methods restored at checkout.
v0.6.8	2026-07-22	Offline Ed25519 license token verifier. Cryptography dependency. Auto qLDPC BP-OSD routing. SparseBlossom MWPM parity fix. BPOSD wall-clock cap fix. OpenCL health check bugfix.
v0.6.7	2026-07-22	Maintenance release published and superseded by v0.6.8; upgrade directly to 0.6.8.
v0.6.6	2026-07-22	Critical import bug fix (removed dead OpenCL reference in v0.6.5). Releases v0.6.3, v0.6.4, v0.6.5 yanked on PyPI.
v0.6.0 - v0.6.2	2026-07-20	AVX2 SIMD CPU parallel batching. Rayon Blossom parallelization. DecoderPool multi-process engine. Out-of-core decode_mmap. Workbench orchestrator.

12. Academic Citations & DOI Metadata

When using QECTOR Decoder v3 in academic publications or commercial benchmarks, cite the official Zenodo DOI records:

- **User Manual & Licensing DOI:** 10.5281/zenodo.21363016 (https://doi.org/10.5281/zenodo.21363016)
- **Performance Benchmarks DOI:** 10.5281/zenodo.21339300 (https://doi.org/10.5281/zenodo.21339300)
- **Architecture Whitepaper DOI:** 10.5281/zenodo.21320543 (https://doi.org/10.5281/zenodo.21320543)

BibTeX Citation Entry

```bibtex
@software{lessard2026qector,
  author    = {Guillaume Lessard},
  title     = {{QECTOR Decoder v3}: Rust/Python Quantum Error Correction Decoding Platform},
  year      = {2026},
  version   = {0.6.9},
  publisher = {Zenodo / iD01t Productions},
  url       = {https://www.qector.store},
  doi       = {10.5281/zenodo.21363016},
  orcid     = {0009-0000-3465-3753}
}
```

Copyright (c) 2026 Guillaume Lessard / iD01t Productions · ORCID: 0009-0000-3465-3753